S.Sivakumar* et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.2, Issue No.5, August – September 2014, 1202 – 1206.

Research Article

Canonical Matching of Tree Structures in Relational Database Management System Using Graph Mining Technique

S. SIVAKUMAR
Research Scholar
MS University, Tirunelveli

Dr.S.P.VICTOR
HOD-CS
St.Xavier's College

Abstract-**The Graph matching offers a convenient way to study structured graph with different level of implications. Our conventional setup initially focuses with a tree and its vertex-edge connectivity. This paper perform a detailed study of classified graph of different propagation towards variant tree in the field of graph mining which can be carried out with isomorphic matching strategies. We will implement our isomorphic tree matching techniques with the implementation in relational database Domains. We will also perform survey analysis strategies for the successful implementation of our proposed research technique in several sampling domains with a maximum level of improvements. In near future we will implement the holographic tree matching in graph mining techniques for predicting the Graph sub structure behaviors.**

Keywords- **Graph Mining, tree matching, search, Database**

I. INTRODUCTION

In graph theory, a tree is an undirected graph in which any two vertices are connected by *exactly one* simple path. In other words, any connected graph without simple cycles is a tree. A forest is disjoint union of trees. The various kinds of data structures referred to as trees in computer science have underlying graphs that are trees in graph theory, although such data structures are generally rooted trees, thus in fact being directed graphs, and may also have additional ordering of branches. Some authors use the term directed rooted tree to refer to rooted trees in their directed graph form, but there is no universally accepted term for this notion; arborescence, out-arborescence, out-tree, and even branching being used to denote the same concept.

A tree is an undirected simple graph G that satisfies any of the following equivalent conditions:

G is connected and has no cycles.

G has no cycles, and a simple cycle is formed if any edge is added to G.

G is connected, but is not connected if any single edge is removed from G.

G is connected and the 3-vertex complete graph K_3 is not a minor of G.

Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

G is connected and has n − 1 edges.

G has no simple cycles and has n − 1 edges.

As elsewhere in graph theory, the order-zero graphs (graph with no vertices) is generally excluded from consideration: while it is vacuously connected as a graph (any two vertices can be connected by a path), it is not 0-connected (or even (−1)-connected) in algebraic topology, unlike non-empty trees, and violates the "one more node than edges" relation.

A leaf is a vertex of degree 1. An internal vertex is a vertex of degree at least 2.

An irreducible (or series-reduced) tree is a tree in which there is no vertex of degree 2.

A forest is an undirected graph, all of whose connected components are trees; in other words, the graph consists of disjoint union of trees. Equivalently, a forest is an undirected cycle-free graph. As special cases, an empty graph, a single tree, and the discrete graph on a set of vertices (that is, the graph with these vertices that has no edges), all are examples of forests.

The term hedge sometimes refers to an ordered sequence of trees.

A polytree (also known as oriented tree or singly connected network) is a directed acyclic graph (DAG) whose underlying undirected graph is a tree. In other words, if we replace its arcs with edges, we obtain an undirected graph that is both connected and acyclic.

A directed tree is a directed graph which would be a tree if the directions on the edges were ignored. Some authors restrict the phrase to the case where the edges are all directed towards a particular

vertex, or all directed away from a particular vertex (see arborescence).

A tree is called a rooted tree if one vertex has been designated the root, in which case the edges have a natural orientation, towards or away from the root. The tree-order is the partial ordering on the vertices of a tree with $u \leq v$ if and only if the unique path from the root to v passes through u. A rooted tree which is a subgraph of some graph G is a normal tree if the ends of every edge in G are comparable in this tree-order whenever those ends are vertices of the tree (16). Rooted trees, often with additional structure such as ordering of the neighbors at each vertex, are a key data structure in computer science; see tree data structure. In a context where trees are supposed to have a root, a tree without any designated root is called a free tree.

In a rooted tree, the parent of a vertex is the vertex connected to it on the path to the root; every vertex except the root has a unique parent. A child of a vertex v is a vertex of which v is the parent.

A labeled tree is a tree in which each vertex is given a unique label. The vertices of a labeled tree on n vertices are typically given the labels 1, 2, n. A recursive tree is a labeled rooted tree where the vertex labels respect the tree order (i.e., if $u < v$ for two vertices u and v, then the label of u is smaller than the label of v).

An n-ary tree is a rooted tree for which each vertex has at most n children. 2-ary trees are sometimes called binary trees, while 3-ary trees are sometimes called ternary trees.

A terminal vertex of a tree is a vertex of degree 1. In a rooted tree, the leaves are all terminal vertices; additionally, the root, if not a leaf itself, is a terminal vertex if it has precisely one child.

There are several approaches to managing graphs in a database. One possibility is to extend a commercial RDBMS engine to support graph structured data. Another possibility is to use general purpose relational tables to store graphs. When these approaches fail to deliver needed performance, recent research has also embraced the challenges of designing a special purpose graph database. Oracle is currently the only commercial DBMS that provides internal support for graph data. Its new 10g database includes the Oracle Spatial network data model [3], which enables users to model and manipulate graph data. The network model contains logical information such as connectivity among nodes and links, directions of links, costs of nodes and links, etc [11]. The logical model is mainly realized by two tables: a node table and a link table, which store the connectivity information of a graph. Still, many are concerned that the relational model is fundamentally inadequate for supporting graph structured data, for even the most basic operations, such as graph traversal, are costly to implement on relational DBMSs, especially when the graphs are large. Recent interest in Semantic Web has spurred increased attention to the Resource Description Framework (RDF) [1]. A triple store is a special purpose database for the storage and retrieval of RDF data. Unlike a relational database, a triple store is optimized for the storage and retrieval of a large number of short statements in the form of subject-predicate-object, which are called triples. Much work has been done to support efficient data access on the triple store [4]. Recently, the semantic web community has announced the billion triple challenges [2], which further highlight the need and urgency to support inferencing over massive RDF data. A number of graph query languages have been proposed since early 1990s [12]. For example, Graph Log [5], which has its roots in Data log, performs inferencing on rules (possibly with negation) about graph paths represented by regular expressions. GOOD [6], which has its roots in object-oriented databases, defines a transformation language that contains five basic operations on graphs [13].

GraphDB [8], another object-oriented data model and query language for graphs, performs queries in four steps, each carrying out operations on subgraphs specified by regular expressions. Unlike previous graph query languages that operate on nodes, edges, or paths, GraphQL [7] operates directly on graphs. In other words, graphs are used as the operand and return type of all operations. GraphQL extends the relational algebraic operators, including selection, Cartesian product, and set operations, to graph structures. For instance, the selection operator is generalized to graph pattern matching. GraphQL is relationally complete and the no recursive version of GraphQL is equivalent to the relational algebra. A detailed description of GraphQL and a comparison of GraphQL with other graph query languages can be found in [9].

With the rise of Semantic Web applications, the need to efficiently query RDF data has been propelled into the spotlight. The SPARQL query language [10] is designed for this purpose [14].

A closely related technique leverages on the substructures in the underlying graphs in order to facilitate indexing. Another way of indexing graphs is to use the tree structures in the underlying graph in order to facilitate search and indexing. The topic of query processing on graph data has been studied for many years, still, many challenges remain. On the one hand, data is becoming increasingly large. One possibility of handling such large data is through parallel processing, by using for example,

the Map/Reduce framework. However, it is well known that many graph algorithms are very difficult to be parallelized. On the other hand, graph queries are becoming increasingly complicated. For example, queries against a complex ontology are often lengthy, no matter what graph query language is used to express the queries. Furthermore, when querying a complex graph (such as a complex ontology), users often have only a vague notion, rather than a clear understanding and definition, of what they query for. These call for alternative methods of expressing and processing graph queries. In other words, instead of explicitly expressing a query in the most exact terms, we might want to use keyword search to simplify queries, or using data mining methods to semi-automate query formation [15].

II. PROPOSED METHODOLOGY

This proposed methodology focuses on the implementation of a Tree Isomorphic algorithmic strategy to search the requested tree details by implementing the matching computations. .

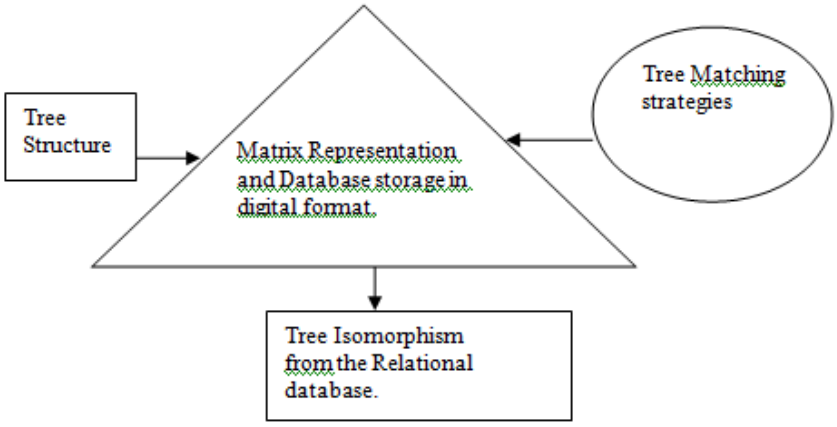

Figure 1.1: Proposed Tree Isomorphic matching structure

III. IMPLEMENTATION

Consider the possible sample tree structures ordered by number of vertices.

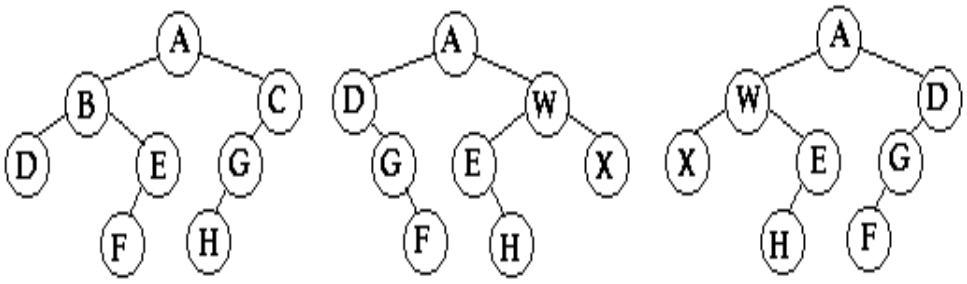

Figure 1.2: Sample Tree Isomorphic matching structures

The tree structure is converted into matrix format as follows,

Step 1: Diagonal values represent the nodes.

Step 2: If the vertex Vi is incident with Vj then edge value=1 else edge value =0

Step 3: Fill the matrix values with edge values accordingly.

Computation

Step 4: Split the Matrix into two parts Upper Triangle and Lower triangle based on the Diagonal nodes and compute Row sum and Column sum respectively.

Step 5: Store the resultant value in a relational database management system.

In our proposed sample tree-1 the matrix value is as follows,

A	1	1	0	0	0	0	0
1	B	0	1	1	0	0	0
1	0	C	0	0	0	1	0
0	1	0	D	0	0	0	0
0	1	0	0	E	1	0	0
0	0	0	0	1	F	0	0
0	0	1	0	0	0	G	1
0	0	0	0	0	0	1	H

In our proposed sample tree-2 the matrix value is as follows,

A	1	1	0	0	0	0	0
1	D	0	1	0	0	0	0
1	0	W	0	0	1	0	1
0	1	0	G	1	0	0	0
0	0	0	1	F	0	0	0
0	0	1	0	0	E	1	0
0	0	0	0	0	1	H	0
0	0	1	0	0	0	0	X

In our proposed sample tree-3 the matrix value is as follows,

A	1	1	0	0	0	0	0
1	W	0	1	1	0	0	0
1	0	D	0	0	0	1	0
0	1	0	X	0	0	0	0
0	1	0	0	E	1	0	0
0	0	0	0	1	H	0	0
0	0	1	0	0	0	G	1
0	0	0	0	0	0	1	F

The computation value using step 4 and step 5 areas follows,

Table 1.1: Tree Computation assessment table

UT-SUM	C1	C2	C3	C4	C5	C6	C7	C8
Tree-1	0	1	1	1	1	1	1	1
Tree-2	0	1	1	1	1	1	1	1
Tree-3	0	1	1	1	1	1	1	1
LTSUM	C1	C2	C3	C4	C5	C6	C7	C8
Tree-1	2	2	1	0	1	0	1	0
Tree-2	2	1	2	1	0	1	0	0
Tree-3	2	2	1	0	1	0	1	0
UT-SUM	R1	R2	R3	R4	R5	R6	R7	R8
Tree-1	2	2	1	0	1	0	1	0
Tree-2	2	1	2	1	0	1	0	0
Tree-3	2	2	1	0	1	0	1	0
LTSUM	R1	R2	R3	R4	R5	R6	R7	R8
Tree-1	0	1	1	1	1	1	1	1
Tree-2	0	1	1	1	1	1	1	1
Tree-3	0	1	1	1	1	1	1	1
	Substring Concatenation Upper and Lower triangle (with a separator *)							
Tree-1	00000000100100011000010001010*1000101110000010000010000000							
Tree-2	00000100000010011000010010100*1001010110000001000000001000							

Tree-3	000000001001000110000100010 1*10001011100000100000100000000

The result is as follows,

Table 1.2: Resultant Tree Isomorphic Details

Substring Compare	Tree Sub string extraction and comparison Values Tree1 and Tree 3 are isomorphic		
Trees	**TREE-1**	Trees	**TREE-1**

IV. CONCLUSION

In this paper, we implemented the tree isomorphic matching technique of graph mining approach with our proposed algorithmic strategy; The tree isomorphism can easily identify the similar structure in any real time domain with different data structure patterns. This research have looked at many of these and discussed their strengths and weaknesses. The overall method proves to be highly efficient compared to mining significant and open trees, dramatically reducing running time and number of features mined. Moreover, the experimental results revealed that the expressiveness of Tree isomorphic node impact influence optimization representatives is significantly higher than that of open trees, because a lower number of features are associated with better accuracy, mainly due to higher specificity, reducing false alarms in matching tasks. In the future this research proposes a Tree-Matcher Implementer for any relational database system entities.

REFERENCES

[1] G. Di Fatta, S. Leue, E. Stegantova. "Discriminative Pattern Mining in Software Fault Detection." Workshop on Software Quality Assurance, 2006.

[2] E.W.Dijkstra. "A note on two problems in connection with graphs. Numerische Mathematik" ,269-271.

[3] T. Falkowski, J. Bartelheimer, M. Spilopoulou. "Mining and Visualizing the Evolution of Subgroups in Social Networks", ACM International Conferenceon Web Intelligence, 2006.

[4] M. Fiedler, C. Borgelt. "Support computation for mining frequent sub graphs in a single graph." Workshop on Mining and Learning with Graphs(MLG'07), 2007.

[5] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Ma-honey. "Statistical properties of community structure in large social and information networks." In WWW, pages 695-704, 2008.

[6] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, J. Zhang. "Graph Distances in the Data-Stream Model". SIAM Journal on Computing, 38(5): pp.1709–1727, 2008.

[7] J. Ferlez, C. Faloutsos, J. Leskovec, D. Mladenic, M. Grobelnik. "Monitoring Network Evolution using MDL". IEEE ICDE Conference, 2008.

[8] F. Eichinger, K. B-ohm, M. Huber. "Improved Software Fault Detection with Graph Mining". Workshop on Mining and Learning with Graphs,2008.

[9] F. Eichinger, K. B-ohm, M. Huber. "Mining Edge-Weighted Call Graphs to Localize Software Bugs". PKDD Conference, 2008.

[10] WFan,K.Zhang, H.Cheng, J. Gao. X. Yan, J. Han, P. S. Yu O. Verscheure.Direct "Mining of Discriminative and Essential Frequent Patterns via Model Search Tree". ACM KDD Conference, 2008.

[11] C. Liu, F. Guo, and C. Faloutsos. Bbm: "Bayesian browsing model from petabyte-scale data. In KDD", pages 537-546, 2009.

[12] Y. Low, J. Gonzalez, A. Kyrola, D. Bick son, C. Guestrin, and J. M. Heller stein. "Graph lab: A framework for parallel machine learning". In UAI, pages 340-349, 2010.

[13] R. Gemulla, E. Nijkamp, P. Haas, and Y. Sisma-nis. "Large-scale matrix factorization with distributed stochastic gradient descent". In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 69-77. ACM, 2011.

[14] A. Ghoting, R. Krishnamurthy, E. P. D. Pednault,B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian,and S. Vaithyanathan. "System: Declarative machine learning on map reduce". In ICDE, pages 231-242, 2011

[15] U. Kang, H. Tong, J. Sun, C.-Y. Lin and C. Faloutsos. "Gbase: an ancient analysis platform for large graphs".VLDB J., 21(5):637-650, 2012.

[16] Diestel, Reinhard (2005), Graph Theory (3rd ed.), Berlin, New York: Springer-Verlag, ISBN 978-3-540-26183-4.

